Berkeley
UNIVERSITY OF CALIFORNIA

# Space Complexity of Minimum Cut Problems in Single-Pass Streams


## Matthew Ding (Berkeley)



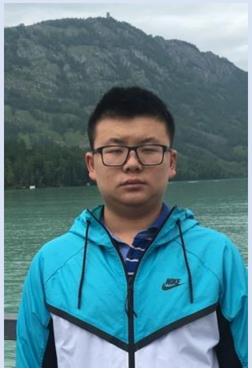
**16th Innovations in Theoretical Computer Science (ITCS 2025)**


Joint work with…

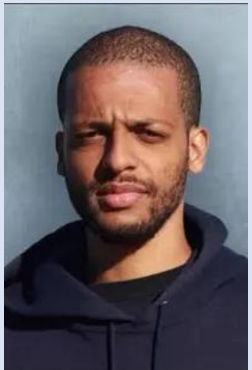

Alexandro Garces (MIT)

Jason Li (CMU)

Honghao Lin (CMU)



Jelani Nelson (Berkeley)



Vihan Shah (Waterloo)

David P. Woodruff (CMU)

# Streaming Algorithms

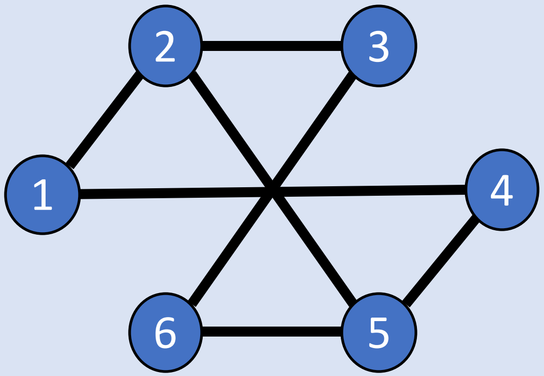
- Data is presented in order one by one over a data stream
  - [Morris 1977] Approximate counting
  - [Flajolet, Martin 1983] Distinct elements
  - [Alon, Matias, Szegedy 1996] Frequency moments

- Logarithmic space with respect to input length

# Graph Streaming

- Graph $G(V, E)$, vertex set V is known
- Edge set E is presented in stream

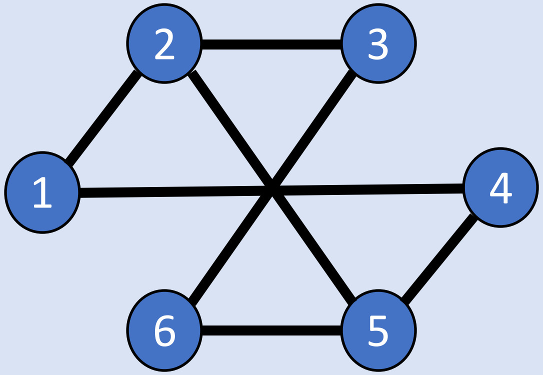
(1, 2)
(4, 5)
(2, 3)
(3, 6)
(5, 6)
(1, 4)
(2, 5)




- Even basic problems require $\Omega(n)$ memory (e.g. connectivity)

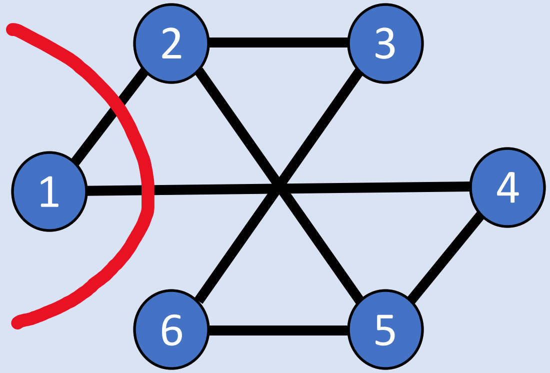
# Graph Semi-Streaming

- Introduced by Feigenbaum, Kannan, McGregor, Suri, Zhang (Theoretical Computer Science 2005)
- $O(n \cdot \mathrm{polylog}\, n)$ space
  - Enough for vertices
  - Not enough for edges



# Minimum Cut Streaming

- Given a graph stream…




find the global minimum cut in the graph.

**Theorem [Zelke 2011]:** Computing the exact minimum cut requires $\Omega(n^2)$ space, i.e., storing all edges graph.

# Minimum Cut Streaming

- How do we deal with this?

I. Approximate minimum cut on weighted graphs in adversarial streams

II. Exact minimum cut on unweighted graphs in random-order streams

- **We show optimal results in both regimes!**

# I. Approximate Minimum Cut in Adversarial Streams

# Cut Sparsification

**Definition (Cut Sparsifier):** $H(V, E')$ is a $(1+\epsilon)$ cut sparsifier of $G(V, E)$ if given a cut S*:*

$$(1-\epsilon)w_G(S, V \setminus S) \leq w_H(S, V \setminus S) \leq (1+\epsilon)w_G(S, V \setminus S)$$

holds **for all** $\emptyset \subseteq S \subseteq V$ with probability $> 2/3$.

- Reweighted subgraph which approximately preserves all cuts

**Theorem [Benczur-Karger 2000]:** Randomized algorithm for $(1+\epsilon)$ cut sparsifiers with $O\left(\frac{n \log n}{\epsilon^2}\right)$ edges

# Spectral Sparsification

**Definition (Spectral Sparsifier):** $H(V, E')$ is a $(1+\epsilon)$ spectral sparsifier of $G(V, E)$ if given a vector $x$:

$$(1-\epsilon)x^\top L_G x \leq x^\top L_H x \leq (1+\epsilon)x^\top L_G x$$

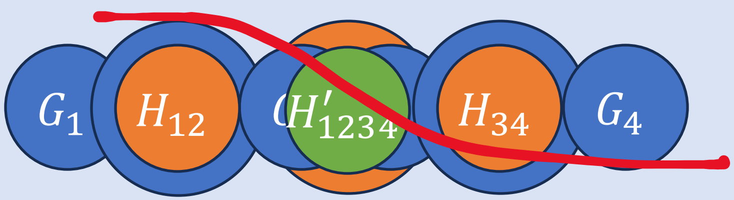
holds **for all** $x \in \mathbb{R}^n$ with probability $> 2/3$.

- Reweighted subgraph whose Laplacian approximately preserves all quadratic forms
- $w_G(S, V \setminus S) = x_S^\top L_G x_S$, where $x_S \in \{0,1\}^n$ is the binary indicator vector of set S

**Theorem [Batson, Spielman, Srivastava 2008]:** Deterministic algorithm for $(1+\epsilon)$ spectral sparsifiers with $O\left(\frac{n}{\epsilon^2}\right)$ edges

# Minimum Cut Streaming

- Merge-and-reduce framework:
  - Merge: Union of two $(1+\epsilon)$ sparsifiers is a $(1+\epsilon)$ sparsifier
  - Reduce: $(1+\epsilon)$ sparsifier of a $(1+\epsilon)$ sparsifier is a $(1+\epsilon)^2$ sparsifier




  - $(1+\epsilon)$-approximate minimum cut (weighted graph, insertion-only stream) with $\tilde{O}(n/\epsilon^2)$ space

# Algorithm Metrics

1. Space Complexity
   - $\tilde{O}(n/\epsilon^2)$ space using merge-and-reduce and Benczur-Karger/Batson-Spielman-Srivastava sparsifiers

2. Running/Update Time
   - $\tilde{O}(m)$ to construct sparsifiers

3. Post-processing/Query Time
   - $\tilde{O}(n/\epsilon^2)$ to find minimum cut on sparsifier

# Algorithm Metrics

1. **Space Complexity**
   - $\tilde{O}(n/\epsilon^2)$ **space using merge-and-reduce and Benczur-Karger/Batson-Spielman-Srivastava sparsifiers**
2. Running/Update Time
   - $\tilde{O}(m)$ to construct sparsifiers
3. Post-processing/Query Time
   - $\tilde{O}(n/\epsilon^2)$ to find minimum cut on sparsifier

# Improving Space Complexity

1. Space Complexity
   - $\tilde{O}(n/\epsilon^2)$ space using merge-and-reduce and sparsifier
   - Any cut sparsifier data structure requires $\Omega(n/\epsilon^2)$ space [Andoni, Chen, Krauthgamer, Qin, Woodruff, Zhang 2015]
   - Introduced "for-each" sparsification

**Definition (For-Each Cut Sparsifier):** $H(V, E')$ is a $(1+\epsilon)$ for-each cut sparsifier of $G(V, E)$ if given a cut S*:*

$$(1-\epsilon)w_G(S, V \setminus S) \leq w_H(S, V \setminus S) \leq (1+\epsilon)w_G(S, V \setminus S)$$

holds **for each** $\emptyset \subseteq S \subseteq V$ with probability $> 2/3$.

# Improving Space Complexity

- **For-each spectral sparsifiers**
  - Constructed with $\tilde{O}(n/\epsilon)$ edges and space, breaking for-all lower-bound of $\Omega(n/\epsilon^2)$
  - Earlier constructions were not graphs, needed to store degrees of vertices
  - Chu, Gao, Peng, Sachdeva, Sawlani, Wang (FOCS 2018): used short-cycle decomposition to preserve degrees

- How do you find the minimum cut?
  - Only polynomial candidate cuts to query (Karger)!

# Algorithm Metrics

1. Space Complexity
   - $\tilde{O}(n/\epsilon^2)$ space using merge-and-reduce and Benczur-Karger/Batson-Spielman-Srivastava sparsifiers

2. Running/Update Time
   - $\tilde{O}(m)$ to construct sparsifiers

3. Post-processing/Query Time
   - $\tilde{O}(n/\epsilon^2)$ to find minimum cut on sparsifier

# Algorithm Metrics

1. Space Complexity (better!)
   - $\tilde{\boldsymbol{O}}(\boldsymbol{n}/\boldsymbol{\epsilon})$ **space using merge-and-reduce and for-each spectral sparsifier [CGPSSW18]**

2. Running/Update Time
   - $\tilde{O}(mn)$ time to run basic short-cycle decomposition

3. Post-processing/Query Time
   - $\tilde{O}(n^3)$ to query all candidate cuts

# Algorithm Metrics

1. Space Complexity (better!)
   - $\tilde{O}(n/\epsilon)$ space using merge-and-reduce and for-each spectral sparsifier [CGPSSW18]

2. Running/Update Time (worse!)
   - **$\tilde{O}(mn)$ time to run basic short-cycle decomposition**

3. Post-processing/Query Time (worse!)
   - **$\tilde{O}(n^3)$ to query all candidate cuts**

# Improving Update Time

2. Running/Update Time
    - $\tilde{O}(mn)$ time to run basic short-cycle decomposition

    - Parter and Yogev (ICALP 2019) give the most recent short-cycle decomposition result
        - Spectral sparsifier with optimal $\tilde{O}(n/\epsilon)$ edges
        - $\boldsymbol{m^{1+o(1)}}$ **total running time**
        - $m^{1+o(1)}$ total working memory

# Improving Update Time

2. Running/Update Time
   - $\tilde{O}(mn)$ time to run basic short-cycle decomposition
   - Parter and Yogev (ICALP 2019) give the most recent short-cycle decomposition result
     - Spectral sparsifier with optimal $\tilde{O}(n/\epsilon)$ edges
     - $\boldsymbol{m^{1+o(1)}}$ **total running time**
     - ~~$m^{1+o(1)}$ total working memory~~
   - We make a slight modification for only $\tilde{O}(m)$ working memory

# Improving Update Time

2. Running/Update Time
   - $\boldsymbol{m^{1+o(1)}}$ **time to run short-cycle decomposition**
   - Can we do even better?
   - **Yes, Online Row Sampling!**

**Theorem [Cohen, Musco, Pachocki 2016]:** Given a graph G over a stream of edges, there exists an online algorithm that constructs a $(1+\epsilon)$ (for-all) spectral sparsifier with $\mathrm{O}(\mathrm{n} \log^2 n / \epsilon^2)$ edges, $\mathrm{O}(n \log^2 n)$ bits of working memory, and $\tilde{O}(m)$ total runtime

- We can turn $m \to \tilde{O}(n/\epsilon^2)$ edges while preserving cuts by a $(1+\epsilon)$ factor

# Algorithm Metrics

1. Space Complexity
   - $\tilde{O}(n/\epsilon)$ space using merge-and-reduce and for-each spectral sparsifier [CGPSSW18]

2. Running/Update Time
   - $\tilde{O}(mn)$ time to run basic short-cycle decomposition

3. Post-processing/Query Time
   - $\tilde{O}(n^3)$ to query all candidate cuts

# Algorithm Metrics

1. Space Complexity
   - $\widetilde{\boldsymbol{O}}(\boldsymbol{n}/\boldsymbol{\epsilon})$ **space using merge-and-reduce and for-each spectral sparsifier instead of** $(\boldsymbol{n}/\boldsymbol{\epsilon})^{\mathbf{1}+\boldsymbol{o}(\mathbf{1})}$ **from [PY19]**

2. Running/Update Time (better!)
   - $\widetilde{\boldsymbol{O}}(\boldsymbol{m}) + (\boldsymbol{n}/\boldsymbol{\epsilon}^{\mathbf{2}})^{\mathbf{1}+\boldsymbol{o}(\mathbf{1})}$ **total time to run [PY19] short-cycle decomposition with online row sampling**
   - If $m > (n/\epsilon^2)^{1+o(1)}$, we get an amortized update time per edge of $\widetilde{\boldsymbol{O}}(\mathbf{1})$ instead of $n^{o(1)}$ from [PY19]

# Approximate Minimum Cut Summary

**Theorem [This work]:** An algorithm calculating $(1+\epsilon)$-approximate minimum cut on weighted graphs in insertion-only streams with

*1.* $\tilde{O}(n/\epsilon)$ *space*
*2.* $\tilde{O}(m) + (n/\epsilon^2)^{1+o(1)}$ *total running time*
*3.* $\tilde{O}(n^2/\epsilon^2)$ *total post-processing time**

1. For-each spectral sparsifier + approximate minimum cut enumeration
2. Improved cycle decomposition + online row sampling

# Approximate Minimum Cut Lower Bounds

- Can we do better?

**Theorem [This work]:** $(1+\epsilon)$-approximate minimum cut algorithms on simple, unweighted graph streams require:

- Randomized: $\Omega(n/\epsilon)$ *space*
- Deterministic: $\Omega(n/\epsilon^2)$ *space (when* $\epsilon \geq 1/n^{1/4}$*)*

- **Algorithm optimal in space complexity (up to polylogarithmic factors)**

# II. Exact Minimum Cut in Random-Order Streams

# Random-Order Model

- Graph is adversarially chosen, a random permutation of the edge set is presented in the stream

- Provable separations:
  - Quantiles: Guha, McGregor (SIAM J. Comput. 2009)
  - Maximum Matching: Bernstein (ICALP 2020)

# Random-Order Minimum Cut

- Can random ordering help beat the $\Omega(n^2)$ lower bound for exact minimum cut algorithms?

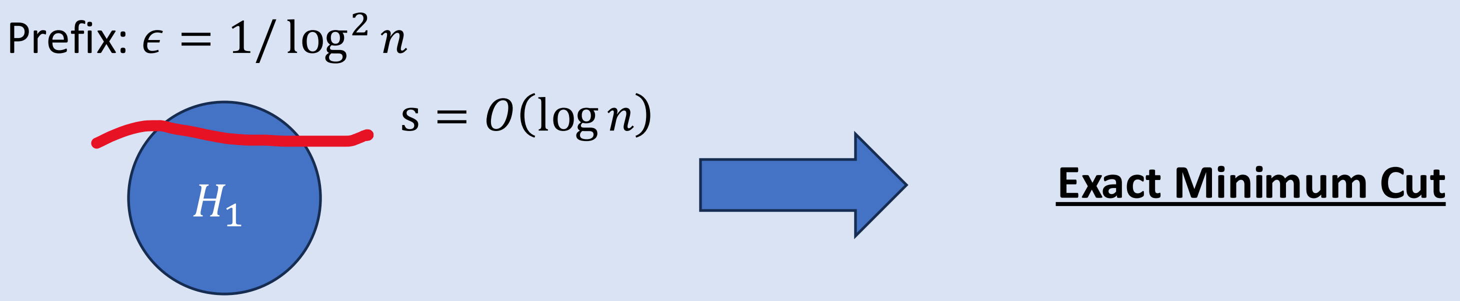
**Theorem [This work]:** There exists an algorithm which computes the exact minimum cut of a simple, unweighted graph in a random-order stream using $\tilde{O}(n)$ space and $\tilde{O}(n)$ update time per edge.

- **Optimal space (up to polylog)!**
  - Chakrabarti, Cormode, McGregor (STOC 2008) showed $\Omega(n)$ space graph connectivity lower bound in random-order streams

# Algorithm Details

- **Key idea: use an initial "prefix" of the edges to get initial information on the graph**

1. Initialize a $\epsilon = 1/\log^2 n$ for-all sparsifier $H_1$, begin inserting edges
   - If minimum cut size $s = O(\log n)$, for-all sparsifier finds exact minimum cut

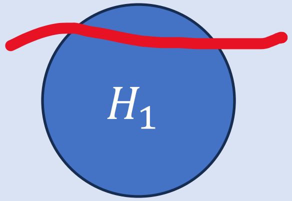


# Algorithm Details

2. Minimum cut size $s = \Omega(\log n)$
    - Assume we know a constant approximation of s (guess powers of 2)
    - **Prefix subgraph** of first $|G| \log n \, / s$ edges constant approximates minimum cut with high probability
    - True minimum cut is a 1.1-approximate minimum cut of prefix graph

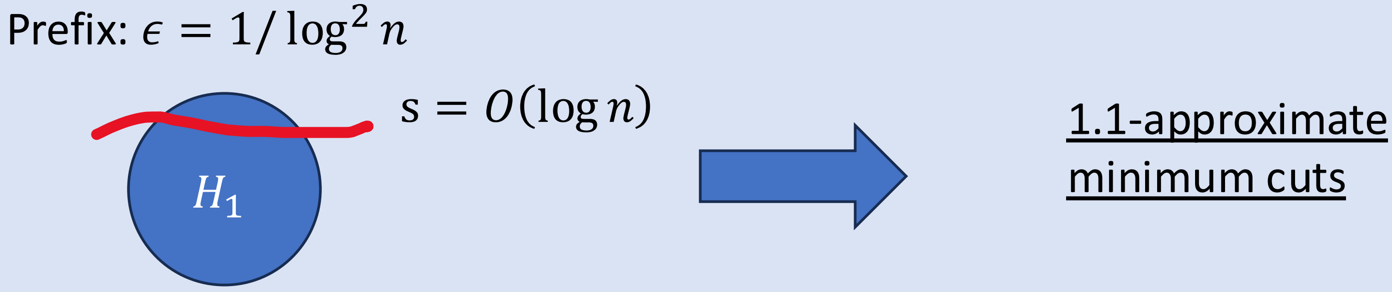
Prefix: $\epsilon = 1/\log^2 n$

$\mathrm{s} = O(\log n)$



# Algorithm Details

2. Minimum cut size $s = \Omega(\log n)$
   - Assume we know a constant approximation of s (guess powers of 2)
   - **Prefix subgraph** of first $|G| \log n \,/s$ edges constant approximates minimum cut with high probability
   - We store $\epsilon = 1/\log^2 n$ sparsifier of **prefix subgraph** to find all candidates: 1.1-approximate minimum cuts



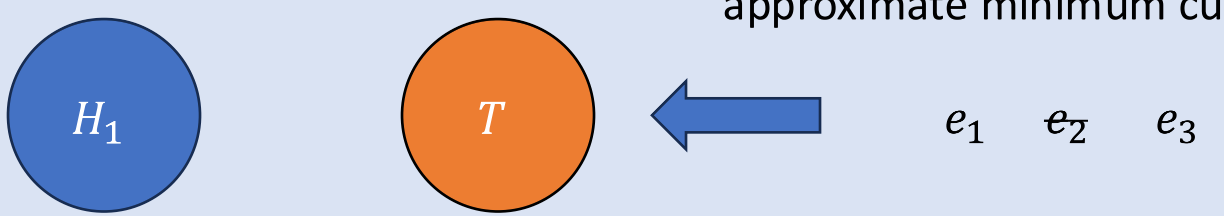
# Algorithm Details

2. Minimum cut size $s = \Omega(\log n)$
   - Remainder of stream: store edges in new graph $T$ only if it is within a non-singleton 1.1-approximate minimum cut
     - Only $O(n)$ edges total, Rubinstein, Schramm, Weinberg (ITCS 2018)
   - Store degrees of vertices (singleton cuts)

Prefix: $\epsilon = 1/\log^2 n$

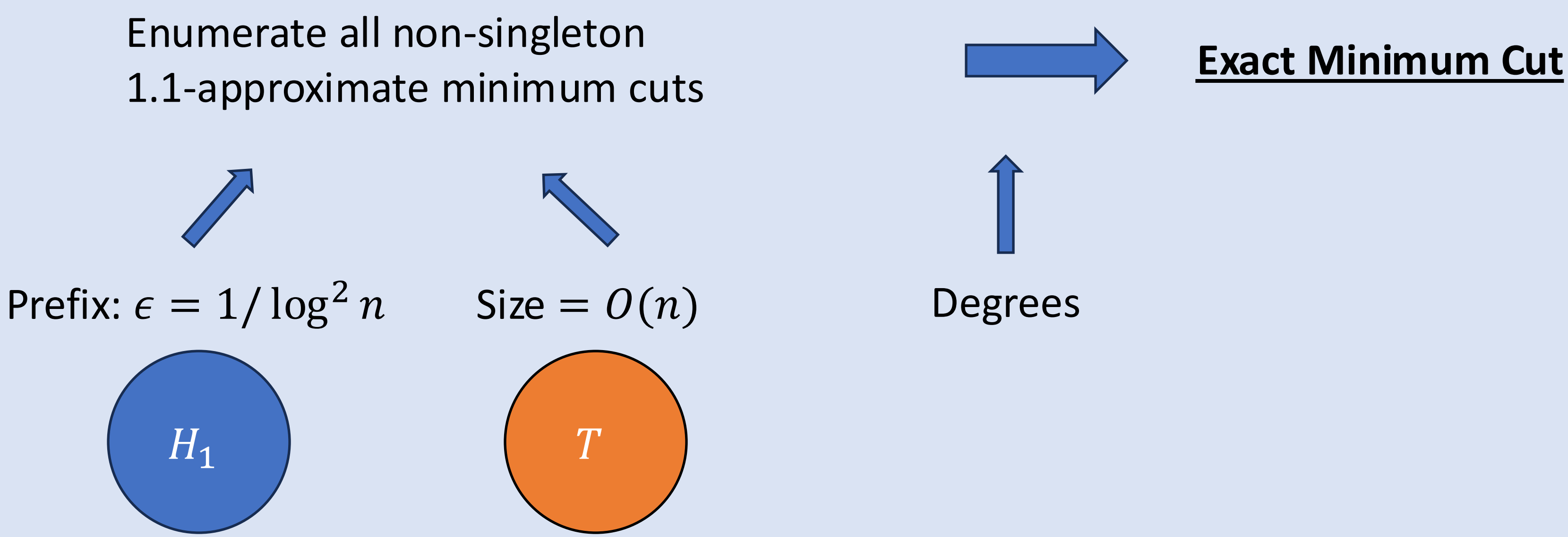
Size $= O(n)$

Edges within non-singleton 1.1-approximate minimum cuts



# Algorithm Details

3. After stream: query all non-singleton approximate minimum cuts
   - Sparsifier $H_1$ gives exact information of prefix
   - Graph $T$ gives exact information on remainder



# III. Improving Update and Post-Processing Times

# Running Time for Approx Minimum Cut

- $\tilde{O}(n^3)$ time to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $O(n)$ time to calculate each cut

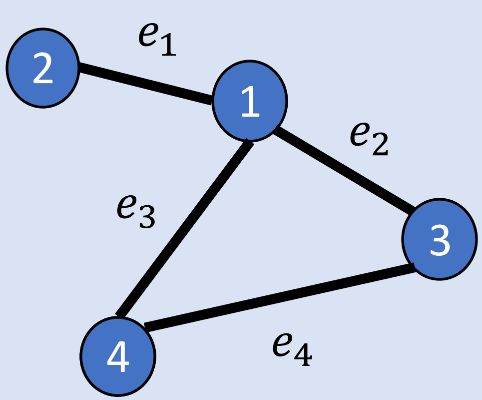
- $w_G(S, V \setminus S) = x_S^\top L_G x_S = x_S^\top B^\top B x_S = \|B x_s\|_2^2$
  - $B \in \mathbb{R}^{\binom{n}{2} \times n}$ is vertex-edge incidence matrix
  - Row for edge e = (u, v) has 1 in column u, −1 in column v, zeroes elsewhere

# Running Time for Approx Minimum Cut

- $\tilde{O}(n^3)$ time to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $O(n)$ time to calculate each cut

$\tilde{O}(n/\epsilon)$ edges

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $e_1$ | 1 | −1 | 0 | 0 |
| $e_2$ | 1 | 0 | −1 | 0 |
| $e_3$ | 1 | 0 | 0 | −1 |
| $e_4$ | 0 | 0 | 1 | −1 |

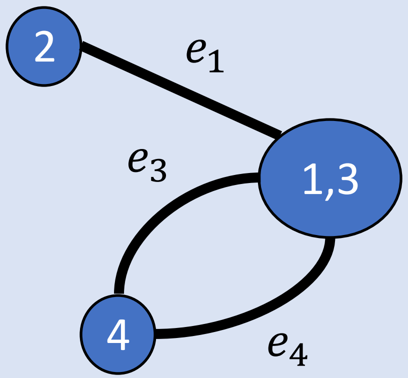


# Running Time for Approx Minimum Cut

- $\tilde{O}(n^3)$ time to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $O(n)$ time to calculate each cut

$\tilde{O}(n/\epsilon)$ edges

| | **1,3** | **2** | **4** |
|---|---|---|---|
| $e_1$ | **1** | **−1** | **0** |
| $e_2$ | **0** | **0** | **0** |
| $e_3$ | **1** | **0** | **−1** |
| $e_4$ | **1** | **0** | **−1** |

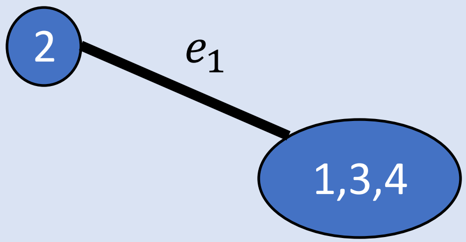


# Running Time for Approx Minimum Cut

- $\tilde{O}(n^3)$ time to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $O(n)$ time to calculate each cut

$\tilde{O}(n/\epsilon)$ edges

| | **1,3,4** | **2** |
|---|---|---|
| $e_1$ | **1** | **−1** |
| $e_2$ | **0** | **0** |
| $e_3$ | **0** | **0** |
| $e_4$ | **0** | **0** |




$$\|Bx_{\{2\}}\|_2^2 = \|Bx_{\{1,3,4\}}\|_2^2 = 1$$

# Running Time for Approx Minimum Cut

- $\tilde{O}(n^3)$ time to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $O(n)$ time to calculate each cut

- Only need $(1+\epsilon)$ approximation…
- **Apply Johnson-Lindenstrauss to vertex-edge incidence matrix!**

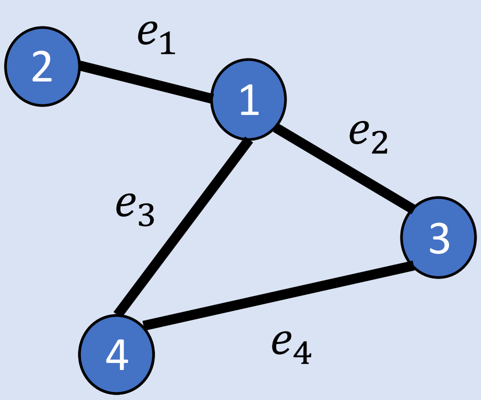
# Running Time for Approx Minimum Cut

- $\tilde{\boldsymbol{O}}(\boldsymbol{n}^2/\boldsymbol{\epsilon}^2)$ to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $\boldsymbol{O}(\log \boldsymbol{n}/\boldsymbol{\epsilon}^2)$ time to calculate each cut

T x

**$\log n/\epsilon^2$ rows**

| | **1** | **2** | **3** | **4** |
|---|---|---|---|---|
| $\boldsymbol{e_1}$ | $\mathbf{1}$ | $\mathbf{-1}$ | $\mathbf{0}$ | $\mathbf{0}$ |
| $\boldsymbol{e_2}$ | $\mathbf{1}$ | $\mathbf{0}$ | $\mathbf{-1}$ | $\mathbf{0}$ |
| $\boldsymbol{e_3}$ | $\mathbf{1}$ | $\mathbf{0}$ | $\mathbf{0}$ | $\mathbf{-1}$ |
| $\boldsymbol{e_4}$ | $\mathbf{0}$ | $\mathbf{0}$ | $\mathbf{1}$ | $\mathbf{-1}$ |



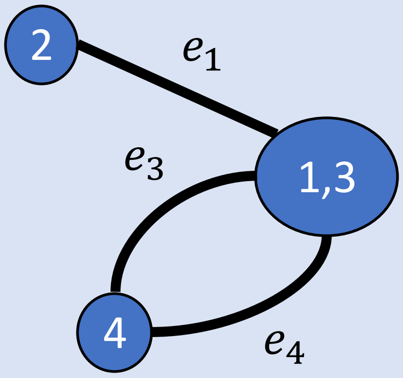
# Running Time for Approx Minimum Cut

- $\tilde{O}(n^2/\epsilon^2)$ to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $O(\log n/\epsilon^2)$ time to calculate each cut

T X

$\log n/\epsilon^2$ **rows**

| | 1,3 | 2 | 4 |
|---|---|---|---|
| $e_1$ | 1 | −1 | 0 |
| $e_2$ | 0 | 0 | 0 |
| $e_3$ | 1 | 0 | −1 |
| $e_4$ | 1 | 0 | −1 |



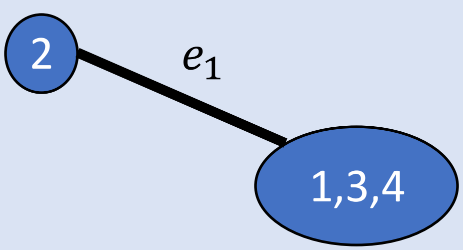
# Running Time for Approx Minimum Cut

- $\tilde{\boldsymbol{O}}(\boldsymbol{n}^2/\boldsymbol{\epsilon}^2)$ to query all candidate cuts in sparsifier using Karger-Stein recursive contraction
  - $\tilde{O}(n^2)$ cuts and $\boldsymbol{O}(\log \boldsymbol{n}/\boldsymbol{\epsilon}^2)$ time to calculate each cut

T X

$\log n/\epsilon^2$ **rows**

| | **1,3,4** | **2** |
|---|---|---|
| $\boldsymbol{e_1}$ | **1** | **−1** |
| $\boldsymbol{e_2}$ | **0** | **0** |
| $\boldsymbol{e_3}$ | **0** | **0** |
| $\boldsymbol{e_4}$ | **0** | **0** |



# Running Time for Exact Minimum Cut

- Runtime Bottlenecks:
  1. Update Time: store edge only if it is within a candidate minimum cut
  2. Post-processing Time: query all candidate minimum cuts with a for-all sparsifier
- Both searches can be improved using a $k$-sparse recovery sketch with $k = \theta(\log n)$!

- **Key idea: applying sketching to Karger-Stein recursive contraction**

# Summary of Results

I. Optimal space approximate minimum cut on weighted graphs in adversarial streams
II. Optimal space exact minimum cut on unweighted graphs in random-order streams
III. General algorithmic framework improving enumerating cuts using sketches

IV. (Optimal space approximate all-pairs effective resistances in adversarial streams)

# Open Problems

- Exact random-order minimum cut on *weighted* graphs

- Approximate minimum cut in *fully-dynamic* graph streams (insertions and deletions):
  - Upper-bound: dynamic spectral sparsifiers $\widetilde{O}\left(\frac{n}{\epsilon^2}\right)$ space

**GAP**

  - Lower Bound: insertion-only minimum cut $\Omega\left(\frac{n}{\epsilon}\right)$ space

# Thank you!

- Full version: **arXiv:2412.01143 [cs.DS]**

- Joint work with:

  **Alexandro Garces, Jason Li, Honghao Lin, Jelani Nelson, Vihan Shah, David P. Woodruff**

- Currently applying for Ph.D. programs this cycle, happy to chat!